Bahasa C++: Konsep Kelas (bagian II)

IF2210 – Semester II 2022/2023

Sumber: Diktat Bahasa C++ oleh Hans Dulimarta

ctor, dtor, dan cctor

- *Constructor/destructor* = method khusus yang secara otomatis dipanggil pada saat penciptaan/pemusnahan objek.
  - Nama konstruktor (ctor) = `NamaKelas`
  - Nama destruktor (dtor) = `~NamaKelas`
- Sebuah kelas memiliki ≥ 0 ctor dan ≤ 1 dtor
- *Copy constructor* (cctor) = konstruktor yang menciptakan objek dengan cara menduplikasi objek lain yang sudah ada
  - Jika tidak dideklarasikan oleh perancang kelas, cctor akan dilakukan secara *bitwise copy*
- Untuk menciptakan array dari objek, kelas objek tersebut harus memiliki *default* ctor.

Constructor

- Tugas utama: menginisialisasi nilai-nilai dari atribut yang dimiliki kelas
- Dua jenis konstruktor:
  - ***Default constructor:*** konstruktor yang menginisialisasi objek dengan nilai(-nilai) default yang ditentukan yang ditentukan oleh perancang kelas. ctor ini **tidak** memiliki parameter formal.
  - ***User-defined constructor***: konstruktor yang menginisialisasi objek dengan nilai(-nilai) yang diberikan oleh pemakai kelas saat objek diciptakan. ctor ini memiliki satu atau lebih parameter formal.

Penciptaan/Pemusnahan Objek

- Beberapa jenis objek dalam program C++:
  - *Automatic object*: diciptakan melalui deklarasi objek di dalam blok eksekusi dan dimusnahkan pada saat blok tersebut selesai eksekusi.
  - *Static object*: diciptakan satu kali pada saat program dimulai dan dimusnahkan pada saat program selesai.
  - *Free store object*: diciptakan dengan operator `new` dan dimusnahkan dengan operator `delete`. Kedua operator dipanggil secara **eksplisit** oleh pemakai.
  - *Member object*: sebagai anggota (atribut) dari kelas.

Contoh Penciptaan/Pemusnahan Objek

```
#include "Stack.h"

Stack s0; /* global (static) */

int reverse() {
  static Stack tStack  = ...; /* local static */
  // kode untuk fungsi reverse() di sini
}

main() {
  Stack s1;        // automatic
  Stack s2 (20);   // automatic
  Stack *ptr;

  ptr = new Stack (50);    /* free store object */
  while (...) {
    Stack s3;   // automatic

    /* assignment dengan automatic object */
    s3 = Stack (5); // ctor Stack(5) is called
    /* dtor Stack(5) is called */

    // ... instruksi lain ...
  } /* dtor s3 is called, just before next iteration,
or before iteration stops */


  delete ptr; /* dtor *ptr is called */
}
/* dtor s2 is called */
/* dtor s1 is called */
```

Catatan: kode program di samping hanya digunakan untuk menggambarkan *lifetime* suatu objek. Dalam program OO sebenarnya, tidak ada variabel atau fungsi global—semua di dalam objek.

Copy Constructor

- *Copy constructor* (cctor) dipanggil pada saat penciptaan objek secara "duplikasi", yaitu:
  - Deklarasi variabel dengan inisialisasi,
    - e.g. `Stack s2 = s1;`
  - *Passing parameter* aktual ke parameter formal secara "pass by value"
  - Pemberian *return value* dari fungsi/method yang nilai kembaliannya bertipe kelas tersebut (bukan ptr/ref)
- cctor untuk kelas X dideklarasikan sebagai

```
X(const X&);
```

```
class Stack {
  public:
    Stack();              // constructor
    Stack (int);          // constructor dengan ukuran stack
    Stack (const Stack&); // copy constructor
    ~Stack();             // destructor
    // ...
};

Stack::Stack (const Stack& s) {
  size = s.size;
  topStack = s.topStack;
  data = new int[size];      // PERHATIKAN: atribut "data"
                             // harus dialokasi ulang,
                             // tidak disalin dari "s.data".
  int i;
  for (i=0; i<topStack; i++) {
    data[i] = s.data[i];
  }
}
```

```
#include "Stack.h"

void f1(const Stack& _) { /* Instruksi tidak dituliskan */ }

void f2(Stack _) { /* Instruksi tidak dituliskan */ }

Stack f3(int) {
   /* Instruksi tidak dituliskan */
   return ...; // return objek bertipe "Stack"
}

main () {
  Stack s2 (20);  // constructor Stack (int)

  /* s3 diciptakan dengan inisialisasi oleh s2 */
  Stack s3 = s2;  // BITWISE COPY, jika
                  // tidak ada cctor yang didefinisikan
  f1(s2);         // tidak ada pemanggilan cctor
  f2(s3);         // ada pemanggilan cctor
  s2 = f3(-100); // ada pemanggilan cctor dan assignment
}
```

ctor Initialization List (1)

- ctor dari atribut akan dipanggil (sesuai urutan deklarasi) sebelum ctor kelas

```
#include "Stack.h"

class Parser {
  public:
    Parser(int);
    // ...
  private:
    Stack sym_stack, op_stack;
    String s;
};
```

- Urutan pemanggilan: ctor `Stack` (2x), ctor `String`, lalu ctor `Parser`. `sym_stack` dan `op_stack` akan diinisialisasi oleh constructor `Stack::Stack()`

ctor Initialization List (2)

- Jika ctor yang diinginkan adalah `Stack::Stack(int)`, maka ctor `Parser::Parser()` harus melakukan *constructor initialization list*

```
Parser::Parser(int x): sym_stack(x), op_stack(x) {
    // ...
}
```

- *Initialization list* dapat berisi ≥ 1 inisialisasi, dipisah koma.
- Setiap inisialisasi mencantumkan nama atribut dengan parameter aktual untuk ctor kelas atribut tsb.

Const Member

- Keyword `const` dapat diterapkan pada atribut maupun method.
- Pada atribut: nilai atribut tersebut akan tetap sepanjang waktu hidup objeknya.
  - Pengisian nilai awal harus dilakukan pada saat objek tersebut diciptakan, yaitu melalui *constructor initialization list*.
- Pada method: method tersebut tidak bisa mengubah (status) data member yang dimiliki oleh kelasnya.
- Object juga dapat ditandai sebagai `const`
  - hanya boleh memanggil const method, untuk memastikan bahwa status object tidak berubah.

Anggota Statik

- Anggota statik adalah anggota yang “dimiliki” oleh kelas, bukan oleh objek dari kelas tersebut.
- Dalam konsep OOP, anggota statik kira-kira adalah atribut & method yang dimiliki oleh objek “kelas”.
  - (Ingat bahwa secara konseptual, kelas pun adalah sebuah objek.)
- Anggota statik juga dapat berupa atribut maupun method.

atribut Statik

```
class Stack {
  public:
    // ... method lain
  private:
    static int n_stack;     // static attribute!!
    // ... atribut & method lain
};
```

- Setiap objek dari kelas memiliki sendiri salinan atribut non-statik
- atribut statik **dipakai bersama** oleh seluruh objek dari kelas tersebut

Inisialisasi Anggota Statik

- Keberadaan anggota statik (method maupun atribut) **tidak** bergantung pada keberadaan objek dari kelas.
- Inisialisasi **harus** dilakukan **di luar** deklarasi kelas dan di luar method. Apa yang terjadi jika diinisialisasi di dalam ctor?
- Dilakukan di dalam file implementasi (X.cc), **bukan** di dalam header file.

```
// inisialisasi atribut statik (file Stack.cc)
int Stack::n_stack = 0;

Stack::Stack() {
  // ... dst
}
```

Method Statik

- Method yang hanya mengakses anggota statik dapat dideklarasikan static di dalam file: Stack.h

```
class Stack {
  // ...
  public:
    static int numStackObj();
  private:
    static int n_stack;
};
```

- Di dalam file: Stack.cc

```
int Stack::numStackObj() {
  // kode mengakses hanya atribut statik
  return n_stack;
}
```

Sifat Anggota Statik

- method statik dapat dipanggil tanpa melalui objek dari kelas tersebut, misalnya:

```
if (Stack::numStackObj() > 0) {
  printf("...");
}
```

- method statik tidak memiliki pointer implisit `this`
- Atribut statik diinisialisasi tanpa perlu adanya objek dari kelas tersebut.

Friend (1)

- `Friend` = pemberian hak pada fungsi/kelas untuk mengakses anggota *non-public* suatu kelas

```
class B { // kelas "pemberi izin"
  friend class A;
  friend void f (int, char *);
}
```

  - *Friend* bersifat satu arah
  - Seluruh member kelas A dan fungsi f dapat mengakses anggota *non-public* dari kelas B

Friend (2)

- Kriteria penggunaan *friend*:
  - Hindari penggunaan *friend* kecuali untuk fungsi operator
  - Fungsi friend merupakan fungsi di luar kelas sehingga objek parameter aktual mungkin dilewatkan secara *call-by-value*
    - Akibatnya operasi yang dilakukan terhadap objek bukanlah objek semula

Nested Class (1)

- Dalam keadaan tertentu, perancang kelas membutuhkan pendeklarasian kelas di dalam deklarasi suatu kelas tertentu
  - Operasi dalam kelas List didefinisikan di kelas List dan mungkin membutuhkan akses kelas ListElem ⟶ kelas List dideklarasikan sebagai friend dari kelas ListElem

```
class List;

class ListElem {
  friend class List;
  public:
    // ...
  private:
    // ...
};

class List {
  public:
    // ...
  private:
    // ...
};
```

Nested Class (2)

- Pemakai kelas `List` **tidak perlu** mengetahui keberadaan kelas `ListElem`
  - Yang perlu diketahui: adanya layanan untuk menyimpan nilai (integer) ke dalam list maupun untuk mengambil nilai dari list
- Kelas `ListElem` dapat dijadikan sebagai *nested class* di dalam kelas `List`

```
class List {
  //
  //
  class ListElem {
    //
    //
  };
};
```

Nested Class (3)

- Di manakah nested class didefinisikan? Di bagian public atau non-public
  - Di bagian public: akan tampak di luar kelas sebagai anggota yang public
  - Di bagian non-public: akan tersembunyi dari pihak luar kelas, tetapi terlihat oleh anggota kelas → efek yang diharapkan

```
class List {
 public:
  // bagian public kelas List
 private:
  class ListElem {
   public:
    // semua anggota ListElem berada pada bagian publik
  }

  // definisi anggota private kelas List
};
```

Method pada Nested Class

- Contoh `List` dan `ListElem`
  - Nama scope yang digunakan untuk kelas `ListElem` adalah `List::ListElem::`, bukan hanya `ListElem::`
    - Jika merupakan kelas generic dengan parameter generic Type, nama scope menjadi `List<Type>::ListElem`

Tugas Baca #3

- “Menulis program berorientasi objek” (8 halaman)
- Buat summary, 1-2 kalimat untuk setiap *heading*.
- Kumpulkan di Edunex